REMARKS

Claims 3, 16 and 29 stand rejected under 35 USC 112 as being indefinite for failing to particularly point out and distinctly claim the subject matter which applicant regards as the invention "Unix-like" which Examiner interpreted as "Multitasking". Claims have responsively been amended to overcome the rejection. A version of the claims, reflecting amendments herein submitted, is set out above. The term "Unix-like" has been deleted. It should be understood that "UNIX" is a registered trademark of The Open Group, but the term as used in the present claims is intended to encompass the other multi-user, general-purpose operating systems mentioned in the present application as being "Unix-like." These include IBM's version of Unix, AIX; HP's version of Unix, HP-UX; and Sun Microsystem's version of Unix, Solaris. Page 6, lines 11-12.

Claims 1-39 stand rejected under 35 USC 103(a). Claims 1, 14 and 27 have responsively been amended to overcome the rejection, except claims 6, 19 and 32 which have been canceled. A version of the claims, reflecting amendments herein submitted, is set out above.

The Office Action acknowledges that in the passage cited, Rosenberg's method for identifying a break point setting involves specifying a file name and line number or file offset in a source file, not in an executable as in the subject of the present application. Applicant agrees. According to the present application "the target is not the process being debugged but the code that is being *executed*." Page 1, line 32 through page 2, line 2. Moreover, Applicant has amended the claims herein to clarify that the combinations of cited references do not teach or suggest the present invention.

Sumi concerns a program development system for simulating execution of a program being debugged, as will be shown herein below. In contrast, the present invention concerns aspects of debugging an executable image in the context of the image's location in a target computer system memory. Page 3, lines 8-14 ("The file, line-number approach is generally used in interactive application debuggers, but depends on the availability of relevant debugging information in the executable image. This approach requires the debugger itself to understand and use this knowledge, to relate the virtual address of the breakpoint to corresponding file name, line number when the breakpoint is hit. Some support from the OS is also needed to achieve this correlation. *Executable images are not generally built with debugging information.* Hence this method cannot be used to identify global breakpoints on generally available executable images.")

(emphasis added); page 7, lines 21-25 ("As shown in Fig. 1, there are for example three processes A, B, and C to which a global breakpoint can be applied. The virtual address space of each process A, B, and C is shown. In each of the virtual address spaces, there is a code segment 102, 104, 106 for memory mapped region of *executable image file* XYZ.SO.") (emphasis added). Notably, since the operating system of a target system already has functionality to deal with the memory context, the debugging in the present invention includes using an operating system module that is the same module available in computer systems that will ultimately execute the executable image, i.e., the "target" computer systems. E.g., page 7, line 31 through page 8, line 10 ("Using the inode and the offset, the Memory Manager module 120 of the operating system kernel locates the physical page where the breakpoint instruction is actually implanted . . . The embodiments have a number of advantages. In particular, the debugger is freed from needing to know anything about the executable format. The method is universally applicable to all types of executables and also applicable to certain executable contexts (for example, shared libraries), which can be loaded at different virtual addresses in different processes. Further, the method does not require any additional debugging information to be present in the executable file.").

Sumi points out a problem of a prior art arrangement that is addressed by his invention, stating that "The disadvantage of such debugging apparatuses which improve the efficiency of program development using correspondence between variables and resources lies in their inability to show the user the content of the optimization executed inside the program conversion unit, so that when the program is greatly rewritten by the internal processing of the program conversion unit, it will take the user a great deal of time to comprehend how his/her program has been rewritten." Col. 3, lines 22-30.

The "rewritten program" in the above passage is referred to elsewhere by Sumi as "execution code." Sumi, col. 16, lines 49-64 ("The code generation unit 110 converts the program rewritten by the optimization unit 106 into execution code and stores the result in the generated code storage unit 103. Here, "execution code" refers to code which can be decoded and executed by the hardware of the target machine. Of special note here is the scale of the target machine, with the program conversion apparatus or the present embodiment generating execution for a system having the hardware model shown in FIG. 7. Here, the central processing unit (CPU) shown in FIG. 7 is fundamentally equipped with the data registers D0, D1, D2, and D3, the address registers A0, A1, A2, and A3, and an arithmetic logic unit (ALU). The CPU is additionally provided with a program counter PC which shows what address is currently being executed, and a stack pointer SP which expresses a starting address in a current stack. Here, the execution code and stack are located in memory.").

While the above passage of Sumi may seem to indicate that this "execution code" is an actual executable image residing in memory of a target computer system, it is clear from other disclosure by Sumi that this is not the case. Sumi sets out FIG. 8A to show an example of the "execution code" sequence stored by the generated code storage unit 103. Sumi, col. 8, lines 10-11. Sumi describes the meaning of the execution code "at each address" in FIG. 8A. Sumi, col. 17, lines 15-16. These addresses are locations in the generated code storage unit 103. Sumi, col. 10, lines 22-28. ("The generated code storage unit 103 is similarly divided into a plurality of small regions, with each of these small regions having a storage region which is assigned an address. These storage regions are used to store execution code when execution code has been generated by the code generation unit 110.").

This generated code storage unit 103 is part of a larger apparatus disclosed by Sumi, a "program development system." Sumi, col. 21, lines 27-35 ("The entire construction of the program development system in the first embodiment is shown in FIG. 4. In this embodiment, the program development system is composed of a program conversion apparatus equipped with a debugging information generation apparatus, a debugging apparatus, a program storage unit 101, a primitive storage unit 102, a generated code storage unit 103, and a debugging information storage unit 104. Here, the information stored by the program storage unit 101, the primitive storage unit 102, the generated code storage unit 103, and the debugging information storage unit 104 is used by both the program conversion apparatus and the debugging apparatus.").

The execution code in the generated code storage unit 103 is processed by a code execution unit 206. Sumi, col. 21, lines 27-35 ("The flowchart in FIG. 13 shows the details of the processing performed by the code execution unit 206 . . . the processing advances to step S72 where the first address "0.times.100" [sic] stored in the generated code storage unit 103 is set in the program counter. Next, in step S73, the code at the address stored in the program counter is fetched from the generated code storage unit 103 . . .").

This code execution unit 206 is *not* the target computer system, and, correspondingly, the "execution code" is not the executable image residing in the memory of the target system. Sumi, col. 21, lines 6-26 ("The code execution unit 206 may be composed of any of a simulator, an incircuit [sic] emulator, or a monitor, so that by using the features which are unique to simulators, incircuit emulators, or monitors, the hardware environment of the target machine can be recreated and a plurality of items of execution code stored in the generated code storage unit 103 can be successively executed in this hardware environment until a breakpoint set by the breakpoint setting unit 203 is reached. . . . while there are differences between the hardware environments of a simulator, an incircuit emulator, and a monitor (while the hardware environments for a monitor or incircuit emulator are the same or very close to that of the real machine, the hardware environment for a simulator is merely a model which is generated on a host computer) all three hardware environments have a common aspect in that they recreate the functions of the data registers D0, D1, D2, and D3, the address registers A0, A1, A2, and A3, the program counter PC, and the stack pointer SP which are shown in the hardware model of FIG. 7.").

Note also that the execution code shown in FIG. 8A of Sumi is in the format of quasi-human-readable sort of machine-language assembly code. The addresses shown are locations of lines in an assembly code listing that include "x" in each "address." See FIG. 8A. Applicant contends that these x'es indicate that it is not known where the memory locations for the lines of a corresponding executable image of the file are actually located in the target system. This is because the focus of Sumi is on a tool to help a developer understand how a break point that the developer places in a line of source code maps to optimized code. That is, to help the developer understand this, the optimized "execution code" is displayed in a manner that the developer can read. Sumi, col. 5, lines 47-56 ("[The system] displays the analyzed details, so that even if the original source code statements have been greatly rewritten, the program developer will soon be able to understand how his/her program has been rewritten during optimization. By doing so, the program developer will not be baffled by the optimization of the program, and will be able to perform the operation verification of the machine language program while conscious of the source code statements written in the high-level programming language.").

In contrast, the present invention concerns an aspect of bridging a gap between a user, who needs to be able to specify a break point in a manner that is easily intelligible to the user, and a debugger that needs to automatically insert, i.e., "represent," the break point in an executable image responsive to the information specified by the user. And it concerns doing this aspect of debugging in a way that is substantially universal, rather than by means of specialized debugger software.

To make more clear the patentable distinction of the present invention, claim 1 and its counterpart independent claims, 14 and 27, have been amended herein. In particular, claim 1 now states, by way of predicate, that a file name is received "for an executable image file," that "the executable image file is loaded in memory of a computer system for execution by the computer system" and that "the global breakpoint is to be placed in the image." Support for this amendment may be found in the present application at page 7, line 17 through page 8, line 2, (describing how the executable relates to the actual physical memory of a target system); see also page 10, line 25 through page 11, line 3 (describing how the memory location is identified when an exception is triggered by a break point instruction at that location). Also, as stated at the beginning of these remarks, the present invention involves debugging the executable image by using operating system modules of a computer system that are the same modules available in the target computer system. That is, the modules are *not* special modules of debugger software. Accordingly, Claim 1 has been amended to state that a symbol expression and the file name are passed "to a first operating system module running on the computer system" and that the symbol expression is "for a location in the executable image file where the global breakpoint is to be placed." Support for this amendment may be found in the present application at page 9, line 29 through page 10, line 5.

The above described amendments more clearly distinguish the present invention from the teaching of Sumi in which execution code is processed by a simulator. In Sumi, the breakpoint setting unit 203 of the development system receives a line indication for a break point from a user and converts this into an address in the execution code. Col. 19, lines 11-15. There is no suggestion by Sumi that this unit 203 includes or uses an operating system module to determine an address, nor that the address that the unit 203 does determine is an actual memory location for an instruction of an executable image loaded in a target computer system.

Similarly, claim 1 and its counterparts have been amended to state that the file name for the executable image file is passed to a second operating system module and that a file offset and file identifier for the executable image are returned by the modules and represented *in* the executable, i.e., inserted into the executable image file. Support for this may be found in the present application at FIG. 3 and page 9, line 25 through page 10, line 15.

Applicant contends that the above amendments to independent claims 1, 14 and 27 overcome the rejections since none of the rejections cite the use of operating system modules to determine actual memory locations for instructions of an executable image loaded in a target computer system and inserting these locations in the executable image. Also, Applicant contends claims 2 through 13, 15 through 26 and 28 through 39 (except canceled claims 6, 19 and 32) are allowable at least due to being dependent upon allowable claims.

PRIOR ART OF RECORD

Applicants have reviewed the prior art of record cited by but not relied upon by Examiner, and assert that the invention is patentably distinct.

REQUESTED ACTION

Applicants contend that the invention as claimed in accordance with amendments submitted herein is patentably distinct, and hereby request that Examiner grant allowance and prompt passage of the application to issuance.

Respectfully submitted,

Anthony V S England

Anthony V. S. England
Attorney for Applicants
Registration No. 35,129
512-477-7165
a@aengland.com